Lógica
Aula 17

Leliane Nunes de Barros

2018

leliane@ime.usp.br

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e
2. Testes caixa-branca: projetados com base no código

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e
2. Testes caixa-branca: projetados com base no código

Análogo a checar:

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e
2. Testes caixa-branca: projetados com base no código

Análogo a checar:

- se uma fórmula proposicional é uma tautologia testando apenas algumas valorações, ou

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e
2. Testes caixa-branca: projetados com base no código

Análogo a checar:

- se uma fórmula proposicional é uma tautologia testando apenas algumas valorações, ou
- se uma fórmula da LPO é um teorema com base na construção de apenas alguns modelos e interpretações.

Teste Versus Verificação

Teste de programas: coleta de evidências sobre a corretude do programa

1. Testes caixa-preta: projetados independentemente do código; e
2. Testes caixa-branca: projetados com base no código

Análogo a checar:

- se uma fórmula proposicional é uma tautologia testando apenas algumas valorações, ou
- se uma fórmula da LPO é um teorema com base na construção de apenas alguns modelos e interpretações.

Um método de teste exaustivo é difícil até para programas pequenos e impossível para programas que podem consumir uma quantidade ilimitada de dados.

Verificação Formal de Programas

Por que verificar?

Verificação Formal de Programas

Por que verificar?

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude
- 10 anos de desenvolvimento

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude
- 10 anos de desenvolvimento
- custo: US$ 7 bilhões

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude
- 10 anos de desenvolvimento
- custo: US$ 7 bilhões
- explodiu em menos de 40 segundos após o lançamento

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude
- 10 anos de desenvolvimento
- custo: US$ 7 bilhões
- explodiu em menos de 40 segundos após o lançamento
- falha no software de cálculo de altitude do sistema de referência inercial.

Ariane 5 (1996)

- foquete lançador descartável usado para colocar satélites artificiais em órbitas geoestacionárias e enviar cargas para órbitas de baixa altitude
- 10 anos de desenvolvimento
- custo: US$ 7 bilhões
- explodiu em menos de 40 segundos após o lançamento
- falha no software de cálculo de altitude do sistema de referência inercial.
- "The internal SRI* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer."

Tipos de verificação

Tipos de verificação

- Verificação baseada em Prova ou em Modelo:

Tipos de verificação

- Verificação baseada em Prova ou em Modelo:
  - Prova: Dado um conjunto de fórmulas lógicas Γ sobre um sistema e uma fórmula φ descrevendo uma propriedade do sistema desejada nessa mesma lógica, queremos encontrar uma demonstração para $\Gamma \vdash \varphi$

Tipos de verificação

- Verificação baseada em Prova ou em Modelo:
  - Prova: Dado um conjunto de fórmulas lógicas Γ sobre um sistema e uma fórmula φ descrevendo uma propriedade do sistema desejada nessa mesma lógica, queremos encontrar uma demonstração para $\Gamma \vdash \varphi$
  - Modelo: Dado que o sistema é representado por um modelo M em uma linguagem apropriada e uma fórmula φ descrevendo uma propriedade desejada, verificar se o modelo satisfaz $M \vDash \varphi$

Tipos de verificação (cont.)

- Verificação baseada em Prova versus Modelo.
- Grau de automatização, sendo os extremos: "completamente automatizado" e "completamente manual". Existem muitas ferramentas no mercado que estão entre esses 2 extremos.

Tipos de verificação (cont.)

- Verificação baseada em Prova versus Modelo.
- Grau de automatização, sendo os extremos: "completamente automatizado" e "completamente manual". Existem muitas ferramentas no mercado que estão entre esses 2 extremos.
- Verificação de propriedade ou sistema completo: verificar uma única propriedade ou o comportamento completo do sistema.

Tipos de verificação (cont.)

- Verificação baseada em Prova versus Modelo.
- Grau de automatização, sendo os extremos: "completamente automatizado" e "completamente manual". Existem muitas ferramentas no mercado que estão entre esses 2 extremos.
- Verificação de propriedade ou sistema completo: verificar uma única propriedade ou o comportamento completo do sistema.
- Domínio de aplicação: hardware ou software; sequencial ou concorrente; programas que terminam ou reativos (que reagem ao ambiente e não se espera que terminem).

Tipos de verificação (cont.)

- Verificação baseada em Prova versus Modelo.
- Grau de automatização, sendo os extremos: "completamente automatizado" e "completamente manual". Existem muitas ferramentas no mercado que estão entre esses 2 extremos.
- Verificação de propriedade ou sistema completo: verificar uma única propriedade ou o comportamento completo do sistema.
- Domínio de aplicação: hardware ou software; sequencial ou concorrente; programas que terminam ou reativos (que reagem ao ambiente e não se espera que terminem).
- Verificação pré versus pós-desenvolvimento: mais vantajosa se introduzida cedo durante o desenvolvimento; erros encontrados depois são mais difíceis e custosos de se detectar.

Motivações para verificação

A verificação de programas serve para:

- **Documentação**: especificações formais servem como base para a escrita do programa

Motivações para verificação

A verificação de programas serve para:

- **Documentação**: especificações formais servem como base para a escrita do programa
- **Redução do tempo de chegar ao mercado**: redução do tempo de desenvolvimento

Motivações para verificação

A verificação de programas serve para:

- **Documentação**: especificações formais servem como base para a escrita do programa
- **Redução do tempo de chegar ao mercado**: redução do tempo de desenvolvimento
- **Reuso**: uma especificação clara e formal permite um melhor reuso de código

Motivações para verificação

A verificação de programas serve para:

- **Documentação**: especificações formais servem como base para a escrita do programa
- **Redução do tempo de chegar ao mercado**: redução do tempo de desenvolvimento
- **Reuso**: uma especificação clara e formal permite um melhor reuso de código
- **Certificados de Auditoria**: sistemas conputacionais de segurança crítica, ou de comércio crítico, demandam que o software seja especificado e verificado com maior rigor e formalidade

Motivações para verificação

A verificação de programas serve para:

- **Documentação**: especificações formais servem como base para a escrita do programa
- **Redução do tempo de chegar ao mercado**: redução do tempo de desenvolvimento
- **Reuso**: uma especificação clara e formal permite um melhor reuso de código
- **Certificados de Auditoria**: sistemas conputacionais de segurança crítica, ou de comércio crítico, demandam que o software seja especificado e verificado com maior rigor e formalidade

A plataforma *A#* é um exemplo de tecnologia emergente que combina verificação de programas, testes e técnicas de verificação de modelos.

O método de verificação de programas que vamos ver: Prova por Triplas de Hoare

Triplas de Hoare: método introduzido por Tony Hoare em 1969.

- Baseado em provas: não verifica exaustivamente todos os estados em que o sistema pode estar (como em Verificação de Modelos, Cap.3), uma vez que é impossível obter modelos finitos com variáveis inteiras;

O método de verificação de programas que vamos ver: Prova por Triplas de Hoare

Triplas de Hoare: método introduzido por Tony Hoare em 1969.

- Baseado em provas: não verifica exaustivamente todos os estados em que o sistema pode estar (como em Verificação de Modelos, Cap.3), uma vez que é impossível obter modelos finitos com variáveis inteiras;
- Semi-automático: regras de cálculo podem ser mecânicas mas alguns passos são manuais;

O método de verificação de programas que vamos ver: Prova por Triplas de Hoare

Triplas de Hoare: método introduzido por Tony Hoare em 1969.

- Baseado em provas: não verifica exaustivamente todos os estados em que o sistema pode estar (como em Verificação de Modelos, Cap.3), uma vez que é impossível obter modelos finitos com variáveis inteiras;
- Semi-automático: regras de cálculo podem ser mecânicas mas alguns passos são manuais;
- Verifica propriedades (e não o comportamento completo do programa)

O método de verificação de programas que vamos ver: Prova por Triplas de Hoare

Triplas de Hoare: método introduzido por Tony Hoare em 1969.

- Baseado em provas: não verifica exaustivamente todos os estados em que o sistema pode estar (como em Verificação de Modelos, Cap.3), uma vez que é impossível obter modelos finitos com variáveis inteiras;
- Semi-automático: regras de cálculo podem ser mecânicas mas alguns passos são manuais;
- Verifica propriedades (e não o comportamento completo do programa)
- Domínio de aplicação: programas sequenciais e "de transformação" (input/output, isto é, não reativos)

O método de verificação de programas que vamos ver: Prova por Triplas de Hoare

Triplas de Hoare: método introduzido por Tony Hoare em 1969.

- Baseado em provas: não verifica exaustivamente todos os estados em que o sistema pode estar (como em Verificação de Modelos, Cap.3), uma vez que é impossível obter modelos finitos com variáveis inteiras;
- Semi-automático: regras de cálculo podem ser mecânicas mas alguns passos são manuais;
- Verifica propriedades (e não o comportamento completo do programa)
- Domínio de aplicação: programas sequenciais e "de transformação" (input/output, isto é, não reativos)
- Usado durante a codificação para pequenos fragmentos.

Etapas da Verificação de Programas

1. Partindo de uma descrição informal R de requisitos do programa, gerar uma fórmula lógica φ_R que especifica formalmente R.

Etapas da Verificação de Programas

1. Partindo de uma descrição informal R de requisitos do programa, gerar uma fórmula lógica φ_R que especifica formalmente R.
2. Escrever o programa P que tem a intenção de satisfazer φ_R (no ambiente de programação usado ou selecionado pelo cliente).

Etapas da Verificação de Programas

1. Partindo de uma descrição informal R de requisitos do programa, gerar uma fórmula lógica φ_R que especifica formalmente R.
2. Escrever o programa P que tem a intenção de satisfazer φ_R (no ambiente de programação usado ou selecionado pelo cliente).
3. Provar $P \vdash \varphi_R$.

Etapas da Verificação de Programas

1. Partindo de uma descrição informal R de requisitos do programa, gerar uma fórmula lógica φ_R que especifica formalmente R.
2. Escrever o programa P que tem a intenção de satisfazer φ_R (no ambiente de programação usado ou selecionado pelo cliente).
3. Provar $P \vdash \varphi_R$.

Etapas da Verificação de Programas

1. Partindo de uma descrição informal R de requisitos do programa, gerar uma fórmula lógica φ_R que especifica formalmente R.
2. Escrever o programa P que tem a intenção de satisfazer φ_R (no ambiente de programação usado ou selecionado pelo cliente).
3. Provar $P \vdash \varphi_R$.

Se o sistema de prova é correto isso implica que o programa satisfaz a sua especificação formal para todas as entradas.

Linguagem de Programação

- Vamos descrever sistemas de prova para programas escritos em uma linguagem simples que contém o núcleo de linguagens como C/C++ e Java.

- A sintaxe do programa é parte do sistema de prova.

Linguagem de Programação

- linguagem imperativa: sequência de comandos, sem concorrência ou *threads*.
- transformacional: a execução do programa transforma um estado inicial (valores iniciais das variáveis do programa) em um estado final.
- contém expressões inteiras e booleanas, atribuições, comandos de seleção (if-then-else), laços (while) e arrays.
- não trataremos elementos comuns às linguagens de programação tais como: funções, objetos, estruturas de dados baseadas em ponteiros ou recursão (esses elementos podem ser implementados em cima da linguagem base, isto é, apartir dos comandos primitivos)

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

- Com entrada $x \geq 0$ o programa devolve $z = 1 + 2 + ... + x$

Exemplo - Soma dos Primeiros x Naturais

```
z = 0;
while (x > 0) {
   z = z + x;
   x = x - 1;
}
```

Queremos poder expressar:

- Com entrada $x \geq 0$ o programa devolve $z = 1 + 2 + ... + x$
- O programa termina.

Como descrever uma especificação φ_R?

Como descrever uma especificação φ_R?

Seja o requisito:

R: "Calcule um número y cujo quadrado seja menor que a entrada x."

Como descrever uma especificação φ_R?

Seja o requisito:

R: "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_R : y.y < x$$

Como descrever uma especificação φ_R?

Seja o requisito:

R: "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_R : y.y < x$$

E se x for -4?

Como descrever uma especificação φ_R?

Seja o requisito:

R: "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_R : y.y < x$$

E se x for -4?

R': "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

Como descrever uma especificação φ_R?

Seja o requisito:

R: "Calcule um número y cujo quadrado seja menor que a entrada x."

$$\varphi_R : y.y < x$$

E se x for -4?

R': "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

Ou seja:

- não basta descrever o estado que deve ser satisfeito
  após a execução do programa !
- também precisamos descrever o estado que
  antecede a execução do programa !

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

φ é a pré-condição

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

φ é a pré-condição

ψ é a pós-condição

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

φ é a pré-condição

ψ é a pós-condição

Notação:

$(|\varphi|)$ é o conjunto de estados que satisfaz φ

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

φ é a pré-condição

ψ é a pós-condição

Notação:

$(|\varphi|)$ é o conjunto de estados que satisfaz φ (pode ser $(|\top|)$)

Triplas de Hoare

Uma especificação é dada por uma tripla

$$(|\varphi|)P(|\psi|)$$

"Se o programa P é executado num estado que satisfaz φ, então o estado resultante da execução de P satisfaz ψ."

φ é a pré-condição

ψ é a pós-condição

Notação:

$(|\varphi|)$ é o conjunto de estados que satisfaz φ (pode ser $(|\mathsf{T}|)$)

$(|\psi|)$ é o conjunto de estados que satisfaz ψ

Triplas de Hoare: exemplo

R: “Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x.”

Triplas de Hoare: exemplo

R: "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

Triplas de Hoare: exemplo

R: "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

```
P:    y = 0
```

ou

Triplas de Hoare: exemplo

R: "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

```
P:    y = 0
```

ou

```
P:    y = 0;
       while (y*y<x) {
          y = y+1;
       y = y-1;
```

satisfazem a especificação! Uma especificação melhor seria:

Triplas de Hoare: exemplo

R: "Se a entrada x é um número positivo, calcule um número y cujo quadrado seja menor que a entrada x."

$$(|x > 0|)P(|y.y < x|)$$

```
P:    y = 0
```

ou

```
P:    y = 0;
       while (y*y<x) {
          y = y+1;
       y = y-1;
```

satisfazem a especificação! Uma especificação melhor seria:

$$(|x > 0|)P(|y.y < x \wedge \forall z(z * z < x \rightarrow z < y)|)$$

Linguagem de especificação e estado do programa

- Fórmulas da LPO com símbolos de funções e de predicados da aritmética inteira:

$$\mathcal{F} = \{-, +, *\} \qquad e \qquad \mathcal{P} = \{<, =\}$$

Linguagem de especificação e estado do programa

- Fórmulas da LPO com símbolos de funções e de predicados da aritmética inteira:

$$\mathcal{F} = \{-, +, *\} \qquad e \qquad \mathcal{P} = \{<, =\}$$

- O Universo de Discurso é o conjunto dos inteiros ($\mathcal{A} = \mathbb{Z}$) e o modelo $\mathcal{M}$ interpreta as funções de $\mathcal{F}$ e predicados de $\mathcal{P}$ de maneira padrão.
- Um estado é uma função I (tabela de contexto):

$$I : Var \longmapsto \mathbb{Z}$$

Linguagem de especificação e estado do programa

- Fórmulas da LPO com símbolos de funções e de predicados da aritmética inteira:

$$\mathcal{F} = \{-, +, *\} \qquad e \qquad \mathcal{P} = \{<, =\}$$

- O Universo de Discurso é o conjunto dos inteiros ($\mathcal{A} = \mathbb{Z}$) e o modelo $\mathcal{M}$ interpreta as funções de $\mathcal{F}$ e predicados de $\mathcal{P}$ de maneira padrão.
- Um estado é uma função I (tabela de contexto):

$$I : Var \longmapsto \mathbb{Z}$$

- Dizemos que um estado I satisfaz φ sse $\mathcal{M}$ satisfaz φ, isto é:

$$I \vDash \varphi \iff \mathcal{M} \vDash_I \varphi$$

Linguagem de especificação e estado do programa

- Fórmulas da LPO com símbolos de funções e de predicados da aritmética inteira:

$$\mathcal{F} = \{-, +, *\} \qquad e \qquad \mathcal{P} = \{<, =\}$$

- O Universo de Discurso é o conjunto dos inteiros ($\mathcal{A} = \mathbb{Z}$) e o modelo $\mathcal{M}$ interpreta as funções de $\mathcal{F}$ e predicados de $\mathcal{P}$ de maneira padrão.
- Um estado é uma função I (tabela de contexto):

$$I : Var \longmapsto \mathbb{Z}$$

- Dizemos que um estado I satisfaz φ sse $\mathcal{M}$ satisfaz φ, isto é:

$$I \vDash \varphi \iff \mathcal{M} \vDash_I \varphi$$

- Dada a tripla $(|\varphi|)P(|\psi|)$, as fórmulas φ e ψ contém apenas quantificadores sobre variáveis que não pertencem ao programa P.

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \Longleftrightarrow \mathcal{M} \vDash_I \varphi$$

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \Longleftrightarrow \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2,\ I(y) = 5,\ I(z) = -1$

(a) $I \vDash \neg(x + y < z)$

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \iff \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2$, $I(y) = 5$, $I(z) = -1$

(a) $I \vDash \neg(x + y < z)$ ✓

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \iff \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2$, $I(y) = 5$, $I(z) = -1$

(a) $I \vDash \neg(x + y < z)$ ✓

(b) $I \vDash y - x * z < z$

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \iff \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2$, $I(y) = 5$, $I(z) = -1$

(a) $I \vDash \neg(x + y < z)$ ✓

(b) $I \vDash y - x * z < z$ **X**

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \Longleftrightarrow \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2$, $I(y) = 5$, $I(z) = -1$

(a) $I \vDash \neg(x + y < z)$ ✓

(b) $I \vDash y - x * z < z$ **X**

(c) $I \vDash \forall u(y < u \rightarrow y * z < u * z)$

Linguagem de especificação e estado do programa: exemplo

$$I \vDash \varphi \Longleftrightarrow \mathcal{M} \vDash_I \varphi$$

Exemplo: $I(x) = -2$, $I(y) = 5$, $I(z) = -1$

(a) $I \vDash \neg(x + y < z)$ ✓

(b) $I \vDash y - x * z < z$ **X**

(c) $I \vDash \forall u(y < u \to y * z < u * z)$ **X**

Notação:

$\vdash_{AR} \varphi \to \psi$ (teorema da aritmética inteira)

Correção Parcial

Relação de satisfação para a correção parcial ($\vDash_{par}$)

$$\vDash_{par} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

"Para qualquer estado satisfazendo φ, se P termina,
o estado final satisfaz ψ".

Qualquer programa que não termina é parcialmente correto.

Correção Total

Relação de satisfação para a correção total ($\vDash_{tot}$)

$$\vDash_{tot} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

"Para qualquer estado satisfazendo φ nos quais P é executado, o programa sempre termina e o estado final satisfaz ψ".

Correção Parcial e Total

$$\vDash_{tot} (|\varphi|)P(|\psi|)$$

$$\Updownarrow$$

$$\vDash_{par} (|\varphi|)P(|\psi|) \text{ e } P \text{ termina.}$$

Em geral, a demonstração da correção total é facilitada fazendo primeiro a demonstração da correção parcial, e em seguida a demonstração que o programa termina.

Linguagem para Descrição de Programas: domínio da aplicação

Expressões inteiras:

$$E ::= n|x|(-E)|(E + E)|(E * E)$$

Linguagem para Descrição de Programas: domínio da aplicação

Expressões inteiras:

$$E ::= n|x|(-E)|(E + E)|(E * E)$$

Expressões booleanas:

$$B ::= \texttt{true}|\texttt{false}|(!B)|(B\&B)|(B||B)|(E < E)$$

$$E_1 == E_2 \equiv !(E_1 < E_2)\&!(E_2 < E_1)$$

Linguagem para Descrição de Programas: domínio da aplicação

Expressões inteiras:

$$E ::= n|x|(-E)|(E + E)|(E * E)$$

Expressões booleanas:

$$B ::= \texttt{true}|\texttt{false}|(!B)|(B\&B)|(B||B)|(E < E)$$

$$E_1 == E_2 \equiv !(E_1 < E_2)\&!(E_2 < E_1)$$

Comandos:

$$C ::= x = E|C; C|\texttt{if } B\{C\}\texttt{else } \{C\}|\texttt{while } B\{C\}$$

Exemplo - Fatorial 1

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```

Exemplo - Fatorial 1

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
z = 0;
while (z != x) {
   z = z + 1;
   y = y * z;
}
```

$$\vDash_{par} (|x \geq 0|) \text{ Fatorial } 1 (|y = x!|)$$

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

$$\vDash_{par} (|x \geq 0|) \text{ Fatorial } 2 (|y = x!|)???$$

Exemplo - Fatorial 2

$$0! = 1$$

$$(n+1)! = (n+1).n!$$

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
}
```

$$\vDash_{par} (|x \geq 0|) \text{ Fatorial 2 } (|y = x!|) \text{ X}$$

$$\vDash_{par} (|x = x_0 \wedge x \geq 0|) \text{ Fatorial 2 } (|y = x_0!|)$$

Variáveis lógicas

São variáveis que só aparecem na especificação φ e não aparecem no programa!

```
Soma:    z = 0;
         while (x > 0) {
           z = z + x;
           x = x - 1;
         }
```

Variáveis lógicas

São variáveis que só aparecem na especificação φ e não aparecem no programa!

```
Soma:    z = 0;
         while (x > 0) {
           z = z + x;
           x = x - 1;
         }
```

$(|x = 3|)$ `Soma` $(|z = 6|)$

Variáveis lógicas

São variáveis que só aparecem na especificação φ e não aparecem no programa!

```
Soma:    z = 0;
         while (x > 0) {
           z = z + x;
           x = x - 1;
         }
```

$(|x = 3|)$ `Soma` $(|z = 6|)$

$(|x = 8|)$ `Soma` $(|z = 36|)$

Variáveis lógicas

São variáveis que só aparecem na especificação φ e não aparecem no programa!

```
Soma:    z = 0;
         while (x > 0) {
           z = z + x;
           x = x - 1;
         }
```

$(|x = 3|)$ `Soma` $(|z = 6|)$

$(|x = 8|)$ `Soma` $(|z = 36|)$

$(|x = x_0 \wedge x \geq 0|)$ `Soma` $(|z = (x_0.(x_0 + 1))/2|)$

Variáveis lógicas

São variáveis que só aparecem na especificação φ e não aparecem no programa!

```
Soma:    z = 0;
         while (x > 0) {
           z = z + x;
           x = x - 1;
         }
```

$(|x = 3|)$ `Soma` $(|z = 6|)$

$(|x = 8|)$ `Soma` $(|z = 36|)$

$(|x = x_0 \wedge x \geq 0|)$ `Soma` $(|z = (x_0.(x_0 + 1))/2|)$

O estado dá um valor para as variáveis do programa, mas não para variáveis lógicas (que podem ser variáveis livres ou variáveis sob o escopo de um quantificador).

Cálculo para Prova de Correção: regra da composição

Regra da Composição

$$\frac{(|\varphi|)\, C_1\, (|\eta|) \qquad (|\eta|)\, C_2\, (|\psi|)}{(|\varphi|)\, C_1; C_2\, (|\psi|)}\ \textit{composição}$$

"Se sabemos que C_1 leva de estados que satisfazem φ em estados que satisfazem η e que C_2 leva de estados que satisfazem η em estados que satisfazem ψ

então

executando C_1 e C_2 (nessa ordem) em estados que satisfazem φ leva a estados que satisfazem ψ."

Cálculo para Prova de Correção: regra da composição

Regra da Composição

$$\frac{(|\varphi|)\, C_1 (|\eta|) \qquad (|\eta|)\, C_2 (|\psi|)}{(|\varphi|)\, C_1 ; C_2 (|\psi|)}\ \textit{composição}$$

O uso numa demosntração é *de baixo para cima*:

para demonstrar

$$(|\varphi|)\, C_1 ; C_2 (|\psi|)$$

precisamos demonstrar

$$(|\varphi|)\, C_1 (|\eta|) \text{ e } (|\eta|)\, C_2 (|\psi|)$$

Cálculo para Prova de Correção: regra da implicação

Regra da Implicação

$$\frac{\vdash_{AR} \varphi' \to \varphi \qquad (|\varphi|)\,C\,(|\psi|) \qquad \vdash_{AR} \psi \to \psi'}{(|\varphi'|)\,C\,(|\psi'|)}\ \textit{implicação}$$

- "Se demonstramos $(|\varphi|)\,C\,(|\psi|)$, fortalecemos a pré-condição φ com o teorema $\vdash_{AR} \varphi' \to \varphi$ e enfraquecemos a pós-condição ψ com o teorema $\vdash_{AR} \psi \to \psi'$, então demonstramos $(|\varphi'|)\,C\,(|\psi'|)$."

Cálculo para Prova de Correção: regra da implicação

Regra da Implicação

$$\frac{\vdash_{AR} \varphi' \to \varphi \qquad (|\varphi|)\, C\, (|\psi|) \qquad \vdash_{AR} \psi \to \psi'}{(|\varphi'|)\, C\, (|\psi'|)}\ \textit{implicação}$$

- "Se demonstramos $(|\varphi|)\, C\, (|\psi|)$, fortalecemos a pré-condição φ com o teorema $\vdash_{AR} \varphi' \to \varphi$ e enfraquecemos a pós-condição ψ com o teorema $\vdash_{AR} \psi \to \psi'$, então demonstramos $(|\varphi'|)\, C\, (|\psi'|)$."
- A regra de demonstração da implicação é importante para completar provas usando lógica de primeira ordem e aritmética de inteiros: faz o elo entre o cálculo de demonstração de programas e a LPO.

Cálculo para Prova de Correção: regra da composição

Regra da Atribuição

$$\frac{}{(|\psi[E/x]|) \quad x = E; \quad (|\psi|)}$$

- Não tem premissas!!
- O uso numa demonstração é *de trás para frente*:
  "Se queremos demonstrar que ψ é verdadeira no estado após a atribuição $x = E$, precisamos mostrar que $(|\psi[E/x]|)$ é verdadeira no estado anterior à execução da atribuição."

Cálculo para Prova de Correção: regra da composição

Regra da Atribuição

$$\frac{}{(|\psi[E/x]|) \quad x = E; \quad (|\psi|)}$$

- Não tem premissas!!
- O uso numa demonstração é *de trás para frente*:
  "Se queremos demonstrar que ψ é verdadeira no estado após a atribuição $x = E$, precisamos mostrar que $(|\psi[E/x]|)$ é verdadeira no estado anterior à execução da atribuição."
- Exemplos:
  $(|2 = y|) \quad x = 2; \quad (|x = y|)$

Cálculo para Prova de Correção: regra da composição

Regra da Atribuição

$$\frac{}{(|\psi[E/x]|) \quad x = E; \quad (|\psi|)}$$

- Não tem premissas!!
- O uso numa demonstração é *de trás para frente*:
  "Se queremos demonstrar que ψ é verdadeira no estado após a atribuição $x = E$, precisamos mostrar que $(|\psi[E/x]|)$ é verdadeira no estado anterior à execução da atribuição."
- Exemplos:
  $(|2 = y|) \quad x = 2; \quad (|x = y|)$

  $(|x + 1 + 5 = y|) \quad x = x + 1; \quad (|x + 5 = y|)$

Cálculo para Prova de Correção: regra da composição

Regra da Atribuição

$$\frac{}{(|\psi[E/x]|) \quad x = E; \quad (|\psi|)}$$

- Não tem premissas!!
- O uso numa demonstração é *de trás para frente*:
  "Se queremos demonstrar que ψ é verdadeira no estado após a atribuição $x = E$, precisamos mostrar que $(|\psi[E/x]|)$ é verdadeira no estado anterior à execução da atribuição."
- Exemplos:
  $(|2 = y|) \quad x = 2; \quad (|x = y|)$

  $(|x + 1 + 5 = y|) \quad x = x + 1; \quad (|x + 5 = y|)$
  que usando a regra da implicação fica:
  $(|x + 6 = y|) \quad x = x + 1; \quad (|x + 5 = y|)$

Prova de Correção: árvore de prova

Existem 2 maneiras diferentes de construção de uma prova de um programa:

1. Como uma árvore de prova, na qual cada nó é rotulado como uma tripla de Hoare. A raiz da árvore (invertida) é rotulada com a especificação que queremos provar; as arestas que saem de cada nó, apontam para os nós filhos de um nó rotulados com os antecedentes da regra de prova usada, para estabelecer a validade da Tripla de Hoare associada aquele nó.

Prova de Correção: prova sequencial

2. Como uma prova sequencial, em que P é visto como uma sequencia de comandos primitivos (atribuições, seleção, repetição):

$$C_1; C_2; ...C_2;$$

assim, uma prova de correção de P para sua especificação:

$$(|\varphi_0|)P(|\psi_n|)$$

pode ser expressa, intercalando fórmulas:

$$(|\varphi_0|), (|\varphi_1|), ..., (|\varphi_n|)$$

com os comandos primitivos::

$$(|\varphi_0|)C_1; (|\varphi_1|)C_2; (|\varphi_2|)...(|\varphi_{n-1}|)c_n; (|\varphi_n|)$$

tal que podemos provar a validade de cada Tripla de Hoare:

$$(|\varphi_i|)C_{i+1}; (|\varphi_{i+1}|).$$